Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ НА ОСНОВІ ОС ANDROID

Текстова частина до курсової роботи
за спеціальністю „Інженерія програмного забезпечення” 121

Керівник курсової роботи
с.в. _Борозенний С.О.__
(прізвище та ініціали)
_____
(підпис)
“____” _____ 2022 р.

Виконав студент _____
___Джосан В.О._____
(прізвище та ініціали)
“____” _____ 2022 р.

Київ 2022

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри мультимедійних систем,
доцент, к.ф-м.н.
_____О. П. Жежерун (підпис)
„____”_____2022 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Джосану Володимиру Олеговичу факультету інформатики 3-го курсу

ТЕМА Розробка мобільного додатку на основі ОС Android

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Вступ

1 Аналіз предметної області

2 Архітектура застосунку та використані технології

3 Інтерфейс застосунку та опис функціоналу

Висновки

Список літератури

Додатки (за необхідністю)

Дата видачі „___” _____ 2021 р. Борозенний О.С. _____ (підпис)

Завдання отримав _____ (підпис)

Календарний план виконання роботи

Тема: Розробка мобільного додатку на основі ОС Android

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	18.10.2021	
2.	Аналіз матеріалів за темою.	Листопад 2021	
3.	Розробка дизайну застосунку.	Грудень 2021	
4.	Розробка застосунку.	Січень – Травень 2022	
5.	Написання текстової частини роботи.	28.05.2022 – 05.06.2022	
6.	Захист курсової роботи.	07.06.2022	

Джосан В. О. _____

Борозенний С. О. _____

“_____”_____

Зміст

Анотація

Метою курсової роботи є створення зручного та функціонального мобільного додатку, який допоможе людям дисциплінувати себе та свою рутину. За допомогою додатку, користувачі зможуть відчути себе у грі, в якій необхідно запрацьовувати кредити (наприклад, виконуючи завдання або утримуючись від певних поганих звичок), за які можна купити винагороди або час на певний вид відпочинку (що визначається самим користувачем).

Застосунок розроблявся в IDE Android Studio та написаний на мові програмування Kotlin.

Ключові слова: розробка застосунку, Android, Kotlin, дисциплінованість, MVVM, Firebase, Firebase Authentication, Cloud Firestore, Cloud Storage, Room, Dagger 2, Navigation Component, Coroutines, Glide, Compressor, DataStore.

Вступ

Проблема дисциплінованості в повсякденному житті була і буде актуальною завжди. Більше того, можна без перебільшення сказати, що сьогодні кожна людина має свій телефон завжди при собі. Комунікація, відпочинок, перегляд новин, – усе це та багато інших звичних занять людей більшою мірою сьогодні активно відбуваються саме в різних застосунках на телефоні, оскільки це зручно.

Саме тому, за мету курсової роботи була поставлена розробка мобільного додатку, який буде корисний людям для збільшення щоденної продуктивності та випрацювання дисциплінованості.

Основна частина роботи складається з 3 розділів:

- Аналіз предметної області: чому саме важливо розвивати дисциплінованість в сучасному світі та як мобільний застосунок може цьому допомогти.
- Розгляд архітектури застосунку, а також використаних бібліотек та технологій.
- Інтерфейс користувача та опис усіх можливостей, які розроблений застосунок надає користувачам.

Основна частина

1. Аналіз предметної області

1.1. Аналіз проблеми дисциплінованості в сучасному світі

Проблема дисциплінованості завжди буде важливою у житті кожної людини. З самого дитинства в кожному з нас батьки та суспільство намагаються виховати цю якість, адже зазвичай саме вона є ключем для досягнення висот в житті, становлення успішною людиною.

Спочатку ми ходимо в дитячий садок, потім в школу, університет та на роботу. Кожна з цих стадій вимагає від людини певних зусиль та неабиякої дисципліни для виконання тих чи інших задач, які, можливо, не дуже хочеться робити. Проте зазвичай людина не може просто відмовитись від своїх зобов’язань та очікувати, що це не вплине негативно на її майбутнє. В школі, не виконавши домашнє завдання, учень отримає погану оцінку, через що, в результаті, він отримає менший бал в атестаті. Це, відповідно, знизить його шанси на поступлення в кращий ВНЗ. В університеті, не підготувавшись до екзамену, студент отримає незадовільну оцінку, через що може втратити бюджетне місце, або ж, навіть, “вилетіти” з університету. Працівник, якщо не виконає призначене йому завдання, або зробить його неякісно, може втратити можливість кар’єрного росту в компанії, або ж взагалі втратити роботу.

На перший погляд, усі згадані вище ситуації, можуть здатись перебільшенням. Проте, дозволивши собі один раз полінуватись (без моментального негативного ефекту або покарання), людина перестає бачити в цьому проблему та продовжує лінуватись у майбутньому, що і призводить до тих неприємних ситуації, описаних вище.

Більше того, зараз, у зв’язку з пандемією, яка охопила весь світ, школи та університети впровадили дистанційне навчання, а більшість компаній забезпечили можливість працювати своїм робітникам з дому. Хоч ситуація в світі зараз стала легшою з появою вакцин, у людей все рівно залишається можливість виконувати свою роботу дистанційно, щоб не наражати себе та своїх рідних на небезпеку. Через це люди почали майже весь свій час проводити вдома, графік став не таким строгим як раніше, а також, через роботу в домашній атмосфері, з’явилась можливість кожну вільну хвилину відволікатись на будь-які інші справи. Усе це сильно вплинуло на дисциплінованість людей та їх ефективність в роботі.

1.2. Android застосунок як вирішення проблеми дисциплінованості

У кожної людини є багато цілей, задач, які необхідно виконати за певний період часу. Це можуть бути як маленькі рутинні справи, які не займають багато часу, так і більш складні задачі, для виконання яких необхідно прикласти досить багато зусиль. Завжди тримати усіх їх в пам’яті – досить непросто, а для того, щоб успішно виконувати всі справи, потрібно завжди слідкувати за тим, щоб правильно розподіляти свій час між роботою та відпочинком. Також у більшості людей є погані звички, які забирають надто багато часу, який краще було б потратити на щось корисне. Саме тому метою курсової роботи була поставлена розробка такого мобільного додатку, який би вирішив усі вищезгадані проблеми.

Чому саме мобільний додаток, адже можна записувати усі свої задачі в блокноті? Відповідь на це запитання – зручність. Сьогодні, для кожної людини телефон – це невід’ємна частина її життя. Неймовірно велика кількість звичних занять людей відбувається саме в мобільних додатках: спілкування з рідними та друзями в соціальних мережах, перегляд новин, використання електронних документів та банківських карт замість матеріальних аналогів, перегляд розважального контенту (відео на YouTube та різноманітні ігри), прослуховування музики та інші. Саме тому, мобільний застосунок – це найкраще рішення.

Як уже згадувалось раніше, розроблений мобільний додаток дозволяє людям відчути себе у грі. У багатьох звичайних іграх, користувач повинен накопичувати ігрову валюту для покупок нових речей або можливостей. У цьому додатку використовується саме такий формат: необхідно вчасно виконувати свої задачі та утримуватись від поганих звичок, щоб отримати кредити, які можна тратити на певні винагороди.

Формат “гри” вибраний не випадково, оскільки саме в іграх у людей зазвичай з’являється азарт до перемоги та здобування певних досягнень.

Звісно, для того, щоб застосунок приніс користь користувачу та допоміг йому стати більш дисциплінованим, він повинен сам вирішити, що необхідно почати удосконалюватись. Адже, за тим, наскільки правдиво користувач виконує завдання та наскільки добросовісно дотримується (утримується від) певних звичок, слідкує він сам. Проте, якщо мотивація розвиватись у користувача дійсно присутня, цей мобільний додаток стане для нього зручним та надійним інструментом.

У додатку користувачі можуть створювати:

- списки задач та самі задачі, з можливістю вибрати дедлайн для їх виконання.
  (Для кожної задачі користувач може визначати кількість кредитів, яку отримає у разі успішного виконання, та штраф, якщо не встигне виконати справу до кінця визначеного терміну).
- звички, яких необхідно позбутись, або ж звички, які потрібно ввести в свою рутину.
  (Для кожної звички користувач може визначати кількість кредитів, яку буде отримувати за кожен день успішного виконання, та штраф, у разі порушення умов певної звички).
- винагороди, які можна купити за накопичені кредити.
  (У якості винагороди, користувач може вибрати будь-що: похід в кіно, прогулянку з друзями, 2 години гри в комп’ютерні ігри, покупку будь-якої речі в магазині. Це може бути будь-яка активність, чи річ, якою користувач готовий себе винагородити в реальному житті).

Додаток підтримує два режими: офлайн та онлайн – в обидвох з яких доступні усі можливості, описані вище. В офлайн режимі немає реєстрації, та вся інформація, створена користувачем, зберігається на одному девайсі. В онлайн режимі для того, щоб користуватись додатком, необхідно зареєструватись за допомогою Google-акаунту. Також з’являється пошук інших користувачів та можливість добавлятись в друзі, що, у свою чергу, дозволяє переглядати звички друзів, яких вони хочуть позбутись. Уся інформація, створена користувачем, зберігається в хмарі, а отже, користувач може доступатись до свого акаунту з різних пристроїв.

Функціональні особливості додатку, описані вище, разом із дружнім інтерфейсом користувача роблять його гарним рішенням для збільшення продуктивності та здобування дисциплінованості в повсякденному житті.

2. Архітектура застосунку та використані технології

2.1. Архітектура застосунку

Визначення з архітектурою застосунку – це одна з найперших і одна з найважливіших задач, над якою повинен задуматись розробник.

За рекомендацією головної документації Android для того, щоб слідувати найкращим практикам розробки Android застосунку, архітектура повинна складатись мінімум із двох (можливо з трьох) рівнів (Рис. 1): UI Layer, Domain Layer (необов'язковий) та Data Layer.

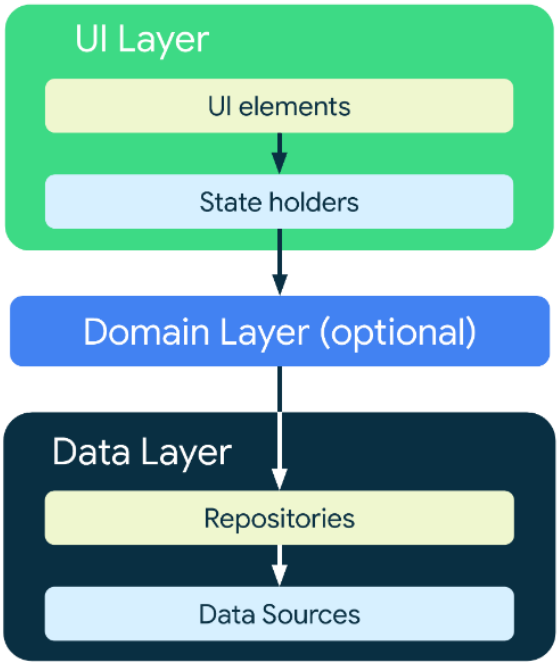


Рис. 1 - Рівні рекомендованої Android розробниками структури застосунку

Роль UI рівня (UI Layer) полягає у відображенні даних програми на екрані. Кожен раз, коли користувач взаємодіє із застосунком, з метою якось змінити дані (наприклад, натиснувши на кнопку) або через будь-які

зовнішні зміни (наприклад, відповідь мережі), інтерфейс користувача має оновлюватися, щоб відобразити ці зміни.

Рівень UI складається з двох підрівнів: UI елементів та зберігачів стану (State Holders). UI елементи – це елементи інтерфейсу користувача (Views), які відображають дані на екрані. State holder'и – це такі класи, елементи архітектури, які зберігають дані, надають їх інтерфейсу та обробляють логіку для їх коректного відображення.

Рівень даних (Data Layer) відповідає за бізнес-логіку програми: логіка створення, зберігання та зміни даних. Data Layer складається із репозиторіїв, кожен з яких може містити декілька джерел даних (data sources). Класи репозиторіїв відповідальні за низку задач: надання даних решті програми, централізація змін даних, прийняття рішень, які джерела даних використовувати в різних ситуаціях, абстрагування джерел даних від решти програми, бізнес логіка програми.

Доменний рівень (Domain Layer) використовується для спрощення та повторного використання взаємодії між інтерфейсом користувача та рівнем даних. Цей рівень відповідає за інкапсуляцію бізнес-логіки, яка повторно використовується кількома класами підрівня State holders (у паттерні програмування це ViewModel). За допомогою цього рівня можна уникнути дублювання коду, розбити великі класи на менші, а також покращити "читабельність" коду та можливості його тестування.

Слідуючи цим рекомендаціям, для розробки застосунку був обраний паттерн програмування MVVM (Model-View-ViewModel), який чудово підходить під усі описані вище архітектурні рішення, запропоновані розробниками Android в документації.

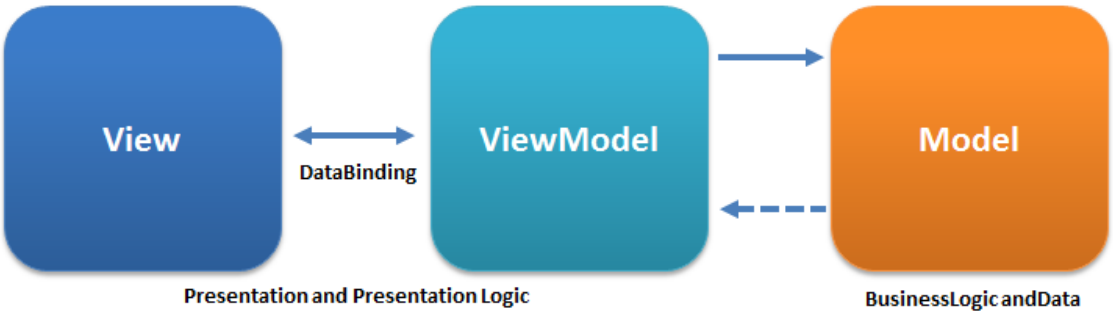


Рис. 2 - Схема паттерну програмування MVVM

Відповідність рівнів паттерну MVVM (Рис. 2) до рівнів рекомендованої архітектури застосунків, розглянутої вище:

- View – підрівень UI elements рівня UI Layer.
- ViewModel – підрівень State holders рівня UI Layer.
- Model – рівень Data Layer.

View – рівень презентації – відповідає за відображення інформації користувачу на екрані. Елементами цього рівня є активності (Activities) та фрагменти (Fragments), які відправляють інформацію відповідно до дій користувача до ViewModel рівня. Проте, згідно паттерну, View рівень не отримує відповіді від ViewModel напряму. Для того, щоб активності чи фрагменти могли дізнатись про зміну даних та відобразити їх на екрані, вони повинні бути підписаними на observable LiveData, що знаходяться в класах ViewModel. При зміні значень LiveData, в активностях та фрагментах, підписаних на них, відбувається callback, в якому і виконується відповідне оновлення UI. Саме така логіка взаємодії View та ViewModel була використана при розробці застосунку.

Model – рівень, що відповідає за бізнес логіку програми та роботу з даними. У розробленому застосунку в якості класів цього рівня були написані та використані репозиторії для кожного типу об’єктів програми

(списки задач, задачі, звички, винагороди, користувачі) та інші репозиторії (для налаштувань застосунку та зображень).

Як було згадано раніше, додаток має два режими: офлайн та онлайн. Для цього у застосунку було визначено декілька джерел даних. В офлайн режимі інформація зберігається на девайсі користувача, а саме в локальній базі даних (за допомогою бібліотеки Room) та internal storage (внутрішньому сховищі застосунку). В онлайн режимі користувачі повинні мати змогу взаємодіяти один з одним, а також мати можливість доступатись до інформації з кількох пристроїв. Для забезпечення такого функціоналу застосунку, була використана платформа Firebase від Google, яка надає низку хмарних послуг та сервісів. Для віддаленої бази даних був використаний сервіс Cloud Firestore – сервіс, що надає інструменти для користування NoSQL базою даних в хмарі, а також сервіс Cloud Storage, за допомогою якого можна зберігати медіа файли в хмарі (детальніше про Firebase та сервіси буде далі). Отже, у застосунку використовуються 4 різних джерела даних, з якими працюють репозиторії.

І останній рівень використаного паттерну MVVM – ViewModel – рівень, що виконує роль моста між View та Model рівнями. Класи цього рівня не мають жодної інформації про те, яка активність чи який фрагмент їх використовує, оскільки ViewModel не зберігає жодного посилання на View класи. Це можливо, оскільки взаємодія в сторону з ViewModel до View відбувається за допомогою LiveData, що було описано раніше. Проте класи ViewModel мають посилання на репозиторії та активно взаємодіють з ними.

2.2. Використані технології та бібліотеки

Firebase — це Backend-as-a-Service (BaaS). Це платформа від Google, що надає розробникам велику кількість інструментів, які допомагають їм розробляти якісні програми. Інструменти, які Firebase надає, охоплюють значну частину сервісів, які розробникам зазвичай доводиться створювати самостійно. Це, наприклад, такі сервіси як аналітика, аутентифікація, бази даних, сховища для зберігання файлів та багато інших. Сервіси розміщуються в хмарі та масштабуються без будь-яких зусиль з боку розробника.

Використані Firebase сервіси:

- **Authentication** (аутентифікація) – надає серверні послуги, прості у використанні SDK і готові UI бібліотеки для автентифікації користувачів у додатку. Підтримується аутентифікація за допомогою паролів, телефонних номерів, аутентифікація за допомогою акаунтів Google, Facebook і Twitter, тощо. Аутентифікація Firebase тісно інтегрується з іншими сервісами Firebase і використовує стандарти, такі як OAuth 2.0 і OpenID Connect.

У розробленому застосунку сервіс використовується в онлайн режимі для аутентифікації користувачів за допомогою Google акаунту.

- **Cloud Firestore –** це сервіс від Firebase і Google Cloud, що надає інструменти для користування гнучкою, масштабованою, NoSQL базою даних в хмарі. Сервіс може синхронізувати дані між клієнтськими додатками за допомогою прослуховувачів (listeners) у реальному часі. Також можна підключити офлайн підтримку

для застосунків, щоб забезпечити для них можливість роботи незалежно від затримки в мережі або підключення до Інтернету. Cloud Firestore також інтегрується з іншими продуктами Firebase і Google Cloud.

Оскільки Cloud Firestore – це NoSQL база даних, яка використовує документи та колекції для зберігання даних, структура сутностей та схема їх відносин були відповідно перенесені без втрати інформації. На верхньому рівні бази даних зберігається колекція користувачів з документами конкретних юзерів. Кожен такий документ містить в собі усю інформацію про користувача та інші колекції з документами, що відповідають наступним сутностям: списки задач, задачі, звички, винагороди. Також ще є 3 колекції з документами з інформацією про друзів користувачів, запити на дружбу та колекція з одним документом, де зберігаються особисті налаштування додатку користувача.

Можливість сервісу оновлювати дані в реальному часі використовується для відображення таких списків: списків задач, задач, звичок та винагород. Для цього були створені класи, які наслідують клас LiveData та реалізовують інтерфейс EventListener. За допомогою цього класу відслідковуються зміни даних в хмарі, записуються як значення LiveData, що у свою чергу викликає callback’и у фрагментах, в яких оновлюються дані для відображення на екрані.

Офлайн підтримку Firestore реалізує за допомогою локального кешування даних. Використання застосунку офлайн (в онлайн режимі, тобто увійшовши за допомогою Google акаунту) використовується лише для перегляду даних. Усі операції на створення або оновлення даних повинні виконуватись з

підключенням до мережі. Рішення таким чином обмежити офлайн функціонал застосунку (в онлайн режимі) було прийняте з наступних причин. Cloud Firestore призначений для використання переважно онлайн, тому створені та оновлені записи, які ще не синхронізовані з сервером, утримуються в черзі. І якщо користувач буде протягом довгого часу виконувати такі операції офлайн, ця черга буде збільшуватись, що сповільнить швидкість виконання усіх запитів (на читання, створення та оновлення).

- **Cloud Storage** – це хмарний сервіс від Firebase, який дозволяє зберігати та отримувати створений користувачами вміст (наприклад, зображення, аудіо чи відеофайли).

У застосунку даний сервіс використовується для зберігання зображень винагород користувачів. Для кожного користувача створюється відповідна тека, яка в назві має його ID, де зберігаються усі його файли. Після завантаження зображення, його посилання зберігається в поле документу відповідної винагороди. І потім, при необхідності відображення картинки винагороди, бібліотека Glide за допомогою посилання на неї виконує дану задачу.

Room – це бібліотека ORM (об’єктно-реляційного відображення). Інакше кажучи, Room зіставляє кортежі сутностей бази даних з об’єктами Java. Room забезпечує рівень абстракції над SQLite, з метою забезпечити більш надійний та вільний доступ до бази даних, при цьому використовуючи всю потужність SQLite.

Бібліотека Room використовується в офлайн режимі застосунку. У проекті визначено 5 сутностей: списки задач (TaskList), задачі (Task), звички (Habit), винагороди (Reward) та користувачі (User). Оскільки мова іде про офлайн режим, в базі даних існує завжди не більше одного

користувача. Сутності TaskList та Task поєднані зв’язком один-до-багатьох (в одному списку задач може бути багато задач). При видаленні списку задач видаляються також усі задачі, які мають зовнішній ключ на нього. Інші дві сутності, Habit та Reward, зберігають в собі інформацію про створені користувачем звички та винагороди відповідно. Поле “imageUri” в сутності Reward – це посилання на зображення винагороди, збережене у внутрішній пам’яті (internal memory) застосунку.

Dagger 2 – фреймворк для ін’єкції залежностей в Java, Kotlin та Android. Він використовує анотації, а також проводить перевірки під час компіляції для аналізу та верифікації залежностей. Для того, щоб добавити залежності в граф, використовуються такі анотації: @Inject (перед конструктором класу, в який необхідно заін’єктити залежності), @Binds (щоб визначити, яку імплементацію інтерфейсу Dagger повинен використовувати) та @Provides (щоб визначити, як Dagger повинен надавати інші класи, які визначені не в нашому проекті).

В проекті був визначений основний компонент (за допомогою анотації @Component), у якого життєвий цикл співпадає з життєвим циклом всього застосунку. Із залежностей, в ньому є applicationContext, а також підключаються 3 модулі, усі з яких надають singleton (на рівні головного компоненту застосунку) залежності: RoomModule – надає екземпляри бази даних Room та DAO усіх сутностей, UtilsModule – надає екземпляр класу, для конвертації різних типів даних, та FirebaseModule – надає об’єкти, для роботи з сервісами платформи Firebase.

Також у головному компоненті визначаються 2 підкомпоненти (@Subcomponent): SignInComponent та MainComponent. У кожної із цих підкомпонент життєвий цикл співпадає із життєвим циклом відповідних активностей (яких у застосунку тільки дві): SignInActivity та MainActivity. У підкомпоненти SignInComponent, окрім унаслідуваних від батьківської

компоненти, є лише одна залежність – це залежність ViewModel'і активності SignInActivity.

Стосовно підкомпоненти MainComponent, вона має залежності усіх ViewModel'ей та фабрик ViewModel'ей фрагментів активності MainActivity, а також залежність, яка визначає, в який режим застосунку користувач увійшов. Значення цієї залежності передається в MainActivity через Intent із SignInActivity (в залежності від вибору користувача) та додається в граф підкомпоненти MainComponent при її створенні. Також до підкомпоненти підключається один модуль – RepositoryModule – він надає залежності реалізацій репозиторіїв, відповідно до вибраного користувачем режиму застосунку (офлайн чи онлайн). Якщо вибраний режим офлайн – будуть використовуватись реалізації репозиторіїв, які працюють з такими джерелами даних, як локальна база даних (Room) та внутрішнє сховище застосунку, інакше реалізації репозиторіїв, які дістають дані з хмари за допомогою сервісів Firebase.

Navigation Component – один із компонентів набору бібліотек Android Jetpack. Даний компонент допомагає більш легко реалізувати навігацію в застосунку.

Navigation Component був використаний в застосунку для налаштування навігації між фрагментами активності MainActivity, оскільки майже увесь функціонал додатку доступний для користувача саме з цієї активності.

Navigation Component надає багато можливостей, деякі з яких були використані в розробленому застосунку: Gradle плагін SafeArgs, за допомогою якого можна передавати типізовані аргументи між фрагментами, а також Bottom Navigation паттерн (меню для переходів між головними фрагментами активності). Також була використана одна з

головних переваг даного компоненту – граф навігації (navigation graph). Для визначення такого графу був створений XML файл, з інформацією про усі фрагменти активності MainActivity, зв’язки (переходи) між ними та аргументи, які передаються при таких переходах. Це дуже зручне рішення, оскільки уся інформація, що стосується навігації, зберігається в одному файлі.

Coroutines – це Kotlin бібліотека, яка надає інструменти для перетворення асинхронних callback'ів для довготривалих задач у послідовний код. Корутини дають можливість керувати тим, як саме та де саме (в яких потоках) повинні виконуватись довготривалі задачі, які інакше можуть заблокувати основний потік і призвести до видимих користувачу “фрізів” (зменшення частоти оновлення дисплею) або ж до блокування взаємодії користувача з додатком. Корутини дозволяють призупиняти виконання коду, продовжуючи його виконання в фоновому потоці, та відновлювати його пізніше в майбутньому, тим самим даючи можливість користувачу безперервно взаємодіяти із застосунком, поки довготривалі задачі виконуються паралельно.

У застосунку корутини використовувались для багатьох задач. В репозиторіях майже усі функції, в яких виконуються будь-які операції з різними джерелами даних, позначені спеціальним словом suspend (яким позначаються функції, які можуть бути призупинені) та при викликах ззовні виконуються в корутинах для того, щоб не блокувати головний потік важкими операціями. У репозиторіях, у випадку виконання запитів до бази даних в хмарі (за допомогою сервісів Firebase) або громіздких операцій з внутрішньою пам’яттю застосунку, у функціях за допомогою диспетчера вручну визначалось, в якому потоці повинен виконуватись код, що це повинні бути саме потоки, виділені під IO операції (які виконуватимуться на фоні). У випадку ж роботи з Room, він сам

піклується про те, щоб виконувати запити не в головному потоці, а отже самому визначати диспетчер не потрібно.

```
override suspend fun updateTaskFinished(task: Task, completed: Boolean, creditsDiff: Long) : Boolean{
    return withContext(Dispatchers.IO){ this: CoroutineScope
        val transaction = fireStore.runTransaction { transaction ->
            val snapshot = transaction.get(tasksRef.parent!!)
            val credits = (snapshot[FBUserConstants.CREDITS] as? Long?) ?: 0L
            transaction.set(tasksRef.document(task.taskId),getTaskAsMap(task.copy(finished = completed)))
            transaction.update(tasksRef.parent!!,FBUserConstants.CREDITS, value: credits + creditsDiff) ^runTransaction
        }
        runCatching { this: CoroutineScope
            Tasks.await(transaction)
            true ^runCatching
        }.getOrElse { it: Throwable
            false
        } ^withContext
    }
}
```

а

```
override suspend fun updateTaskFinished(task: Task, completed: Boolean, creditsDiff: Long) : Boolean{
    tasksDatabaseDAO.update(task.copy(finished = completed))
    if(creditsDiff!=0L){
        usersDatabaseDAO.updateCredits(creditsDiff)
    }
    return true
}
```

б

```
suspend fun updateTaskFinished() : Boolean{
    _showLoadingPanel.value = true
    val result = tasksRepository.updateTaskFinished(task,completed,_credits.value!!)
    _showLoadingPanel.value = false
    if(!result) _showConnectionError.value = true
    return result
}
```

в

```
binding.confirm.setOnClickListener { it: View!
    if(requireActivity().checkForInternet()){
        lifecycleScope.launch { this: CoroutineScope
            val result = viewModel.updateTaskFinished()
            if(result){
                if(args.fromMain) findNavController().navigateUp()
                else findNavController().navigate(TaskFinishedDialogDirections
                    .actionTaskCompletedDialogToTaskDetailsFragment(args.task.copy(finished = args.completed)))
            }
        }
    }
}
```

г

Рис. 3 - Приклад використання корутин в застосунку

Glide — це швидка та ефективна Android бібліотека для керування та завантаження медіа-файлів: кадрів відео файлів, зображень та анімованих GIF-файлів. Glide підтримує декодування медіа-файлів, їх кешування в пам’яті та на диску, а також підтягування їх із ресурсів застосунку. Також одна із переваг даної бібліотеки є простий та зручний інтерфейс.

У застосунку Glide використаний для відображення зображень винагород та аватарів користувачів. В онлайн режимі картинки підтягуються по наданій Glide URL-адресі, а також використовується кешування, для можливості відображення зображення у разі відсутності з’єднання з мережею. В офлайн режимі картинки підтягуються за допомогою URI зображення, збереженого у внутрішній пам’яті застосунку. В обидвох випадках використовуються два допоміжних зображення: анімована картинка загрузки, яка показується, поки завантажується основне зображення, та зображення картинки з помилкою, у разі виникнення якихось помилок.

```
fun ImageView.loadImageFromUriString(uri: String, mode: Mode){
    if(mode == Mode.ONLINE){
        Glide.with(context)
            .load(uri)
            .placeholder(R.drawable.loading_animation)
            .error(R.drawable.ic_broken_image)
            .into( view: this)
    }else{
        Glide.with(context)
            .load(uri)
            .diskCacheStrategy(DiskCacheStrategy.NONE)
            .skipMemoryCache( skip: true)
            .placeholder(R.drawable.loading_animation)
            .error(R.drawable.ic_broken_image)
            .into( view: this)
    }
}
```

Рис. 4 - Приклад використання бібліотеки Glide

Compressor – це легка та потужна Android бібліотека для стиснення зображень. Компресор дозволяє стискати великі фотографії в зображення меншого розміру з дуже незначною втратою в якості.

Дана бібліотека була використана в застосунку для того, щоб зменшувати вагу зображень винагород, з ціллю використання якомога менше пам’яті на телефонах користувачів. Також однією з причин було хмарне сховище від Firebase, адже в даному сервісі існують обмеження на кількість доступного місця в сховищі для кожного проекту, якщо використовувати його безкоштовно. Чим більше місця для медіа-файлів застосунок потребує – тим більше це буде коштувати.

```
val compressedImageFile = Compressor.compress(requireContext(), file) { this: Compression
    resolution( width: 1000, height: 1000)
    quality( quality: 80)
    format(Bitmap.CompressFormat.JPEG)
    size( maxFileSize: 1_048_576)
}
```

Рис. 5 - Приклад використання бібліотеки Compressor

DataStore – це частина набору бібліотек Android Jetpack. Це рішення для зберігання даних, яке дозволяє зберігати пари ключ-значення або типізовані об’єкти. Для роботи використовуються Kotlin корутини та Flow для асинхронного та транзакційного збереження даних. У DataStore є дві реалізації: Preferences DataStore та Proto DataStore. За допомогою першої реалізації можна зберігати дані за допомогою ключів, без забезпечення безпеки типів (type safety). Натомість другий варіант зберігає типізовані об’єкти, при цьому вимагаючи визначення їх схеми.

У проекті був використаний перший варіант реалізації DataStore – Preferences DataStore – для збереження користувацьких налаштувань додатку та інших незалежних налаштувань в обидвох режимах застосунку.

DataStore використовувався як ще одне джерело даних, а отже, для нього був створений репозиторій, який Dagger постачав ViewModel’ям в офлайн режимі. В онлайн моді, у свою чергу, налаштування зберігаються в хмарі, а саме в Firestore.

```
override suspend fun getIntValue(key: String, defaultValue: Int) : Int{
    val preferencesKey = intPreferencesKey(key)
    val data = dataStore.data.first()
    return data[preferencesKey] ?: defaultValue
}

override suspend fun saveIntValue(key: String, value: Int) {
    val preferencesKey = intPreferencesKey(key)
    dataStore.edit { settings ->
        settings[preferencesKey] = value
    }
}
```

Рис. 6 - Приклад використання DataStore

3. Інтерфейс застосунку та опис функціоналу

Перед розробкою андроїд застосунку, з метою спрощення цього процесу, був використаний онлайн сервіс Figma для створення дизайну користувацького інтерфейсу (Рис. 7).

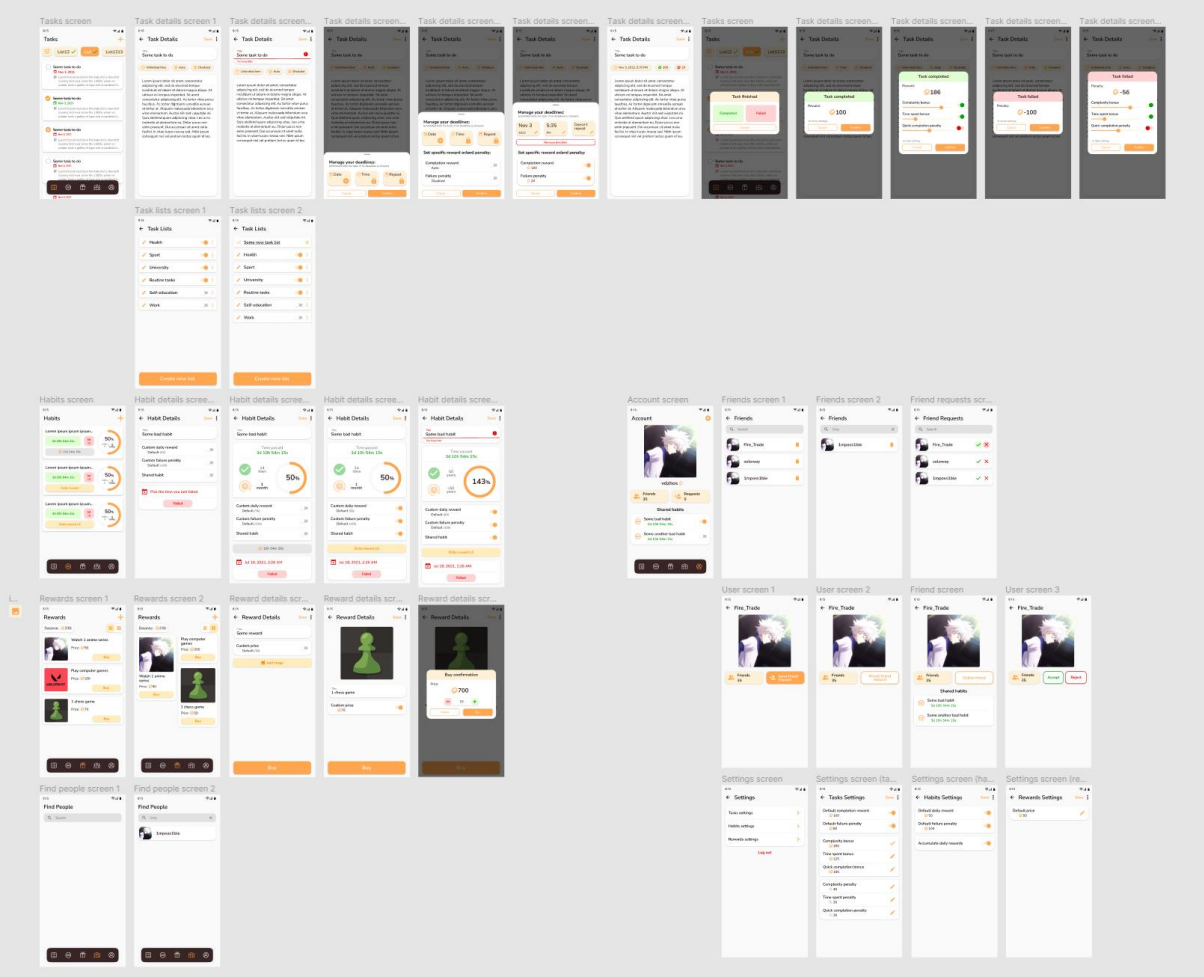


Рис. 7 - Розроблений дизайн застосунку в Figma

Усі описані нижче вікна підтримують обидва режими екрану: портретний та альбомний.

Коли користувач вперше відкриє додаток, він побачить вікно (Рис. 8) із назвою застосунку та двома кнопками для вибору режиму (онлайн чи офлайн). Щоб увійти в онлайн режим, користувачу необхідно натиснути на кнопку з надписом “Sign in with Google”, щоб увійти за допомогою

свого облікового запису Google. Для використання офлайн режиму застосунку потрібно натиснути на нижню кнопку “Start offline”. Обраний користувачем режим запам’ятовується при наступному відкритті застосунку.

Відтепер і надалі до всіх скріншотів інтерфейсу користувача буде добавлена сіра межа, оскільки задній фон застосунку зливається з фоном білого аркушу. Насправді ця межа відсутня в самому додатку.

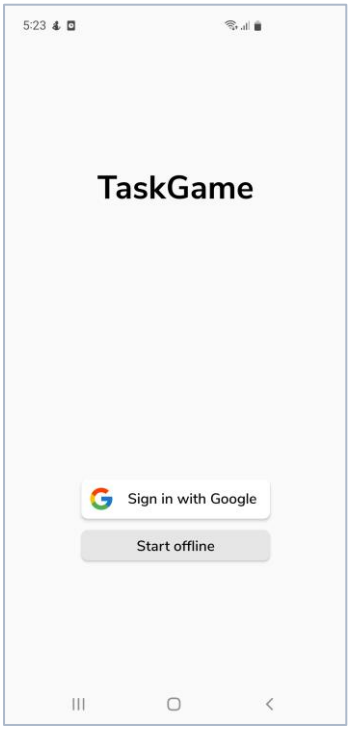
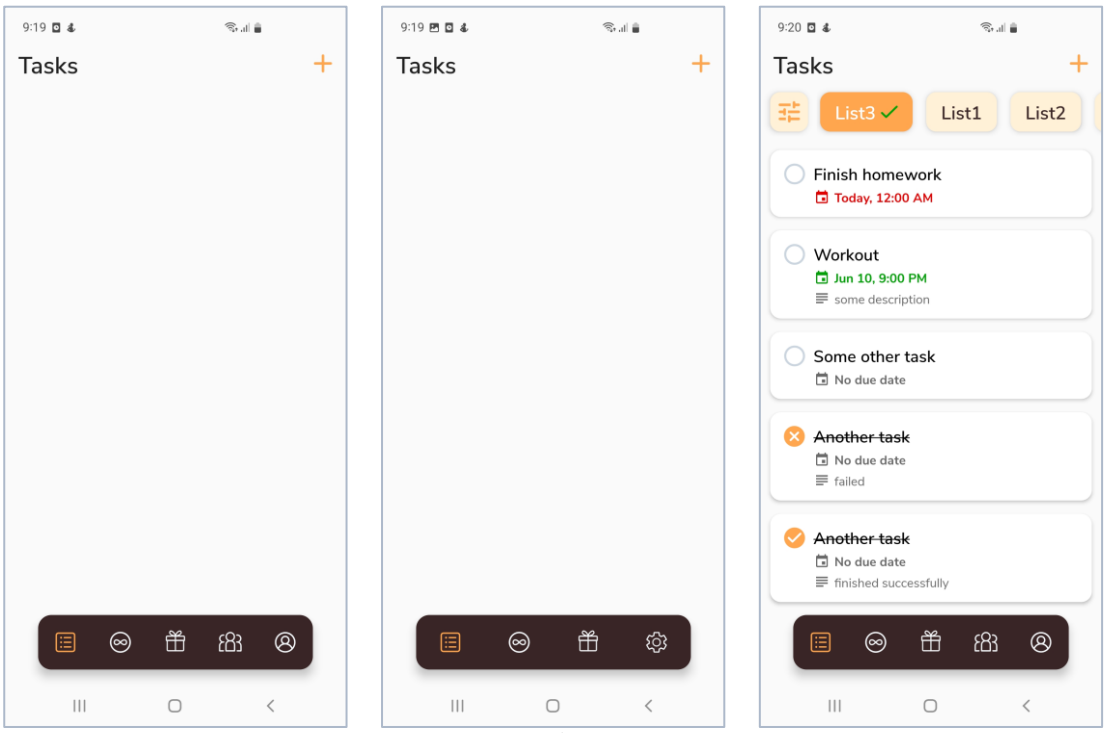


Рис. 8 - Головне вікно застосунку

Після вибору користувачем режиму застосунку, відкривається вікно з усіма списками задач та задачами (Рис. 9). В різних режимах вигляд екрану відрізняється тільки вкладками нижнього навігаційного меню. В онлайн режимі їх 5 (Рис. 9а), в офлайн режимі – 4 (Рис. 9б). Спільні вкладки для двох режимів: задачі, звички, винагороди. В офлайн режимі за останньою вкладкою користувач може перейти до вікна налаштувань, а в онлайн режимі четверта вкладка – перехід до вікна пошуку користувачів, п’ята вкладка – перехід до особистого кабінету.

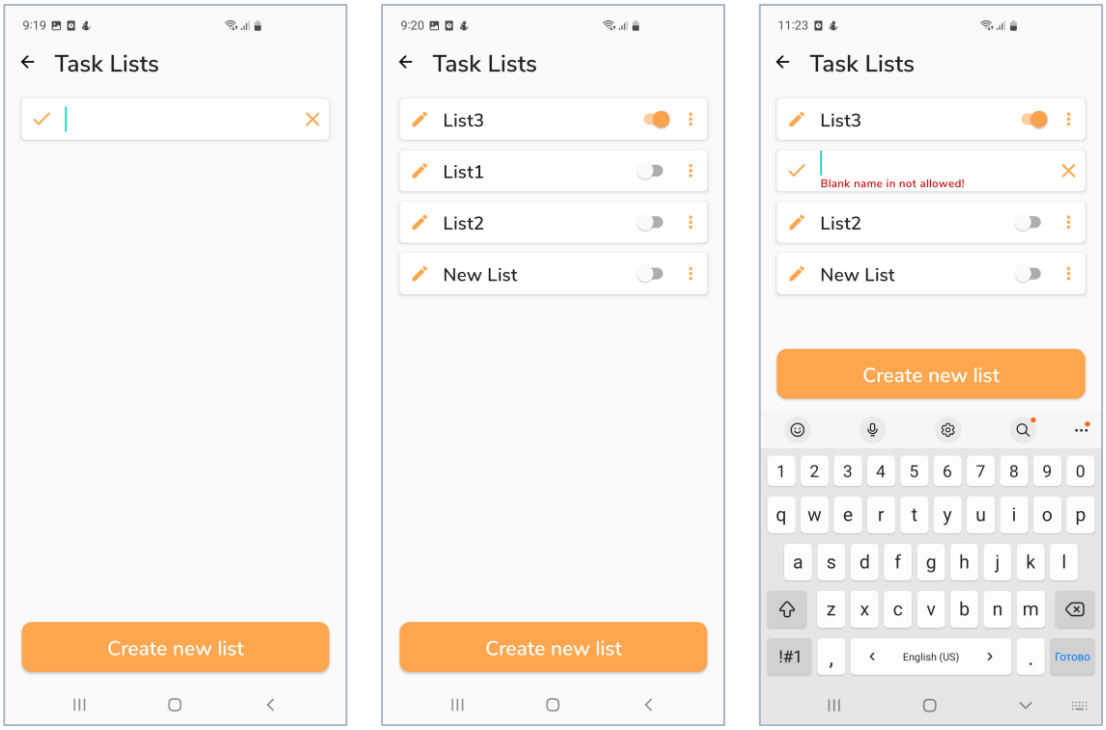
У випадку, якщо у користувача немає жодного створеного списку задач або жоден список задач не вибраний (вибраний список задач підсвічується оранжевим кольором і вибрати його можна натиснувши на нього), за кнопкою “плюс” (на панелі інструментів) він перейде до вікна створення/управління списками задач (Рис. 10) (також до цього вікна можна перейти, натиснувши на кнопку ліворуч від усіх списків задач (Рис. 9в)). Якщо ж якийсь список задач вибраний, користувач перейде до вікна створення нової задачі (Рис. 11а) у вибраному списку. Для того, щоб перейти до вікна редагування задачі (Рис. 11а), користувачу потрібно на неї натиснути. При натисканні на кружечок невиконаної задачі (зліва зверху), відкривається діалогове вікно вибору результату виконання задачі (Рис. 12а). Також у користувача є можливість змінити стан активності списку задач за допомогою подвійного натискання (у активного списку з’являється зелена галочка і він починає відображатись на початку списку).



а б в

Рис. 9 - Вікно зі списками задач (а - онлайн режим, б - офлайн режим, в - онлайн режим із заповненими даними)

На вікні створення/управління списками (Рис. 10) знизу знаходиться кнопка, натиснувши на яку, зверху в списку з’являється новий список задач, який потрібно якось назвати (Рис. 10а). Усі списки задач можна редагувати: змінювати їх назви, стан активності та видаляти (викликавши контекстне меню потрібного списку). Назва списку задач повинна бути унікальною, не бути пустою та бути не більшою ніж 50 символів, інакше користувачу буде виведене відповідне повідомлення з помилкою (Рис. 10в).



а б в

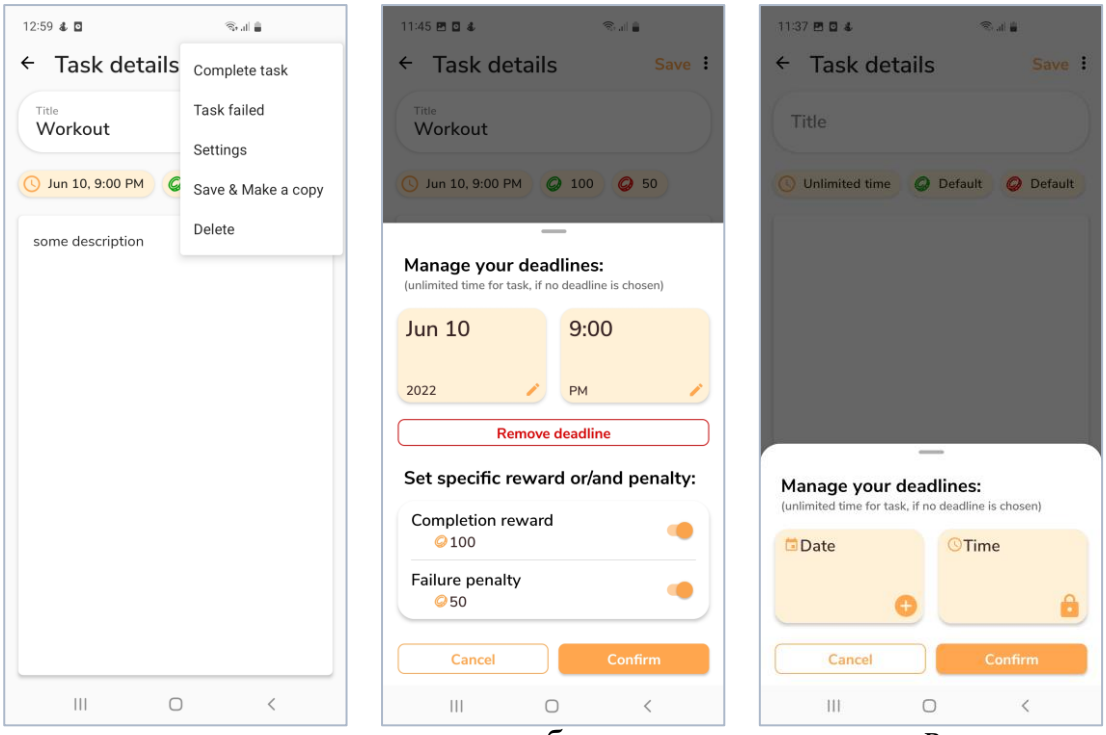
Рис. 10 - Вікно зі списками списків задач (а - створення нового списку, б - вигляд із заповненими даними, в - некоректна назва списку)

На вікні створення/редагування задачі (Рис. 11а) є поле з назвою задачі та описом задачі. Назва повинна бути введена обов’язково та бути не довшою ніж 100 символів, інакше буде виведене відповідне повідомлення з помилкою. Всі решту полей та налаштувань задачі – необов’язкові. Під полем з назвою задачі є 3 пункти (chip’и), в яких

вказана інша інформація про налаштування задачі: дедлайн, кількість кредитів за успішне виконання та невиконання завдання. Натиснувши на пункт з дедлайном задачі, користувачу відкриється вікно (Bottom Sheet) наполовину, тільки з видимими налаштуваннями дедлайну (встановлення дати та часу) (Рис. 11в) . Якщо ж користувач натисне на будь-який пункт з кредитами, вікно з налаштуваннями відкриється повністю (Рис. 11б). Для визначення дедлайну необхідно вказати дату (в діалоговому вікні для вибору дати), і коли вона буде вибрана, у користувача також з’явиться можливість налаштувати час (в діалоговому вікні для вибору часу). Число кредитів, доступне для введення в обидвох полях, повинне бути щонайбільше восьмизначним. Якщо користувач не визначить кредити за успішне виконання або за невиконання завдання, для цієї задачі будуть використовуватись значення з загальних налаштувань для задач (буде далі).

В контекстному меню користувач може:

- Успішно завершити завдання (перехід до відповідного вікна (Рис. 12б)). Ця опція доступна користувачу, якщо задача досі не виконана, та поки термін виконання задачі не закінчився (або якщо дедлайну немає).
- “Провалити” завдання (перехід до відповідного вікна (Рис. 12в)). Ця опція доступна, якщо завдання досі не виконане.
- Відкрити вікно з налаштуваннями задачі (Рис. 11б)
- Зберегти відкриту задачу та створити копію (до назви нової задачі добавляється текст “Copy”)
- Видалити задачу

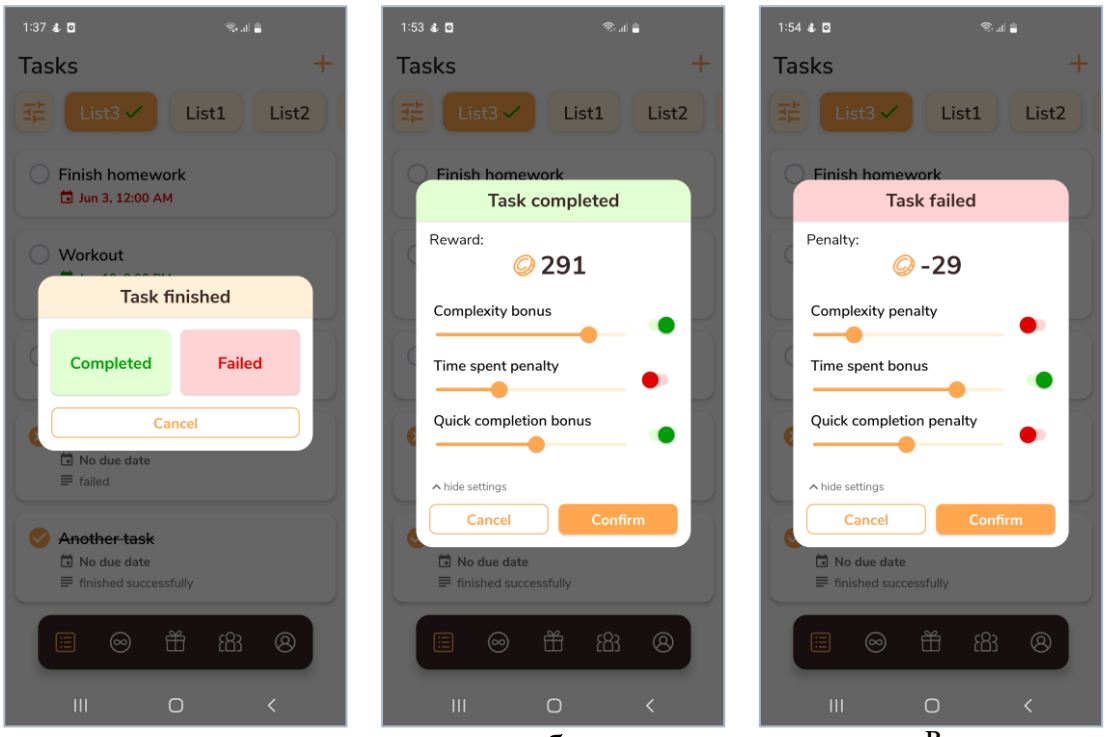


а б в

Рис. 11 - Вікно створення/редагування задачі (а - головне вікно з контекстним меню, б - повне вікно налаштувань, в - неповне вікно налаштувань)

На діалоговому вікні вибору результату виконання завдання (Рис. 12а) є дві кнопки: “Completed”, за якою користувач перейде до вікна успішного виконання задачі (Рис. 12б), та “Failed” – перехід до вікна невиконаного/”проваленого” завдання (Рис. 12в). На обидвох цих вікнах відображається відповідна кількість кредитів, які будуть добавлені до/зняті з рахунку користувача. Проте заздалегідь досить складно передбачити наскільки складним буде завдання, скільки часу воно займе, а також наскільки швидко з’явиться час для його виконання. Саме тому, для того щоб мати змогу відреагувати на ці фактори та відповідно відрегулювати нагороду/штраф за виконання завдання, користувач може використати розширені налаштування: додати бонус або збільшити штраф за цими трьома пунктами. Повзунок для кожного фактору – це відсоток бонусу/штрафу, який застосовується до кінцевої суми кредитів. У

налаштуваннях користувач повинен заздалегідь визначити значення бонусів та штрафів (визначаються у відсотках від винагороди (для бонусів) або від штрафу (для додаткових штрафів) задачі). Наприклад, нехай кількість кредитів за успішне виконання завдання – 100, а бонус за складність завдання – 50%. Якщо за допомогою відповідного повзунка користувач виставить максимальне значення (100%), то до початкової ціни завдання (100 кредитів) буде додана бонусна сума кредитів (100*0.5*1 = 50 кредитів). Якщо ж застосувати половину бонусу, його значення буде дорівнювати 25 кредитам (100*0.5*0.5 = 25).

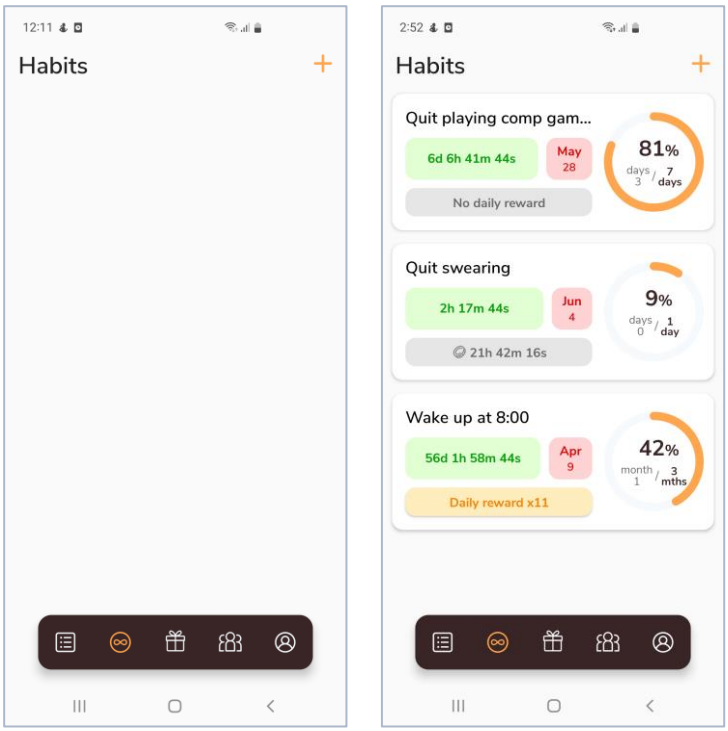


а б в

Рис. 12 – Діалогові вікна виконання задачі (а - вибір результату виконання, б - успішне виконання задачі, в – невиконання задача)

Наступне вікно в нижньому навігаційному меню – список із усіма звичками (Рис. 13). Так само, як і на вікні з усіма задачами, натиснувши на кнопку “плюс” або на конкретну звичку зі списку, користувач перейде до вікна створення/редагування звички (Рис. 14). На кожній звичці у списку відображається інформація про те, скільки часу користувач успішно

утримується від/дотримується певної звички, дата останньої невдачі, скільки часу (у відсотках) пройшло від останньої “круглої дати” до наступної, а також кнопка, натиснувши на яку користувач може забрати винагороду за один день успішного дотримання умов звички (або за більше днів, якщо в налаштуваннях визначено, що нагороди повинні акумулюватись). Якщо ж в самій звичці та в загальних налаштуваннях звичок не визначені кількість кредитів для щоденної винагороди, кнопка буде заблокована та на ній виводитиметься текст, що повідомляє про відсутність щоденної винагороди “No daily reward”.

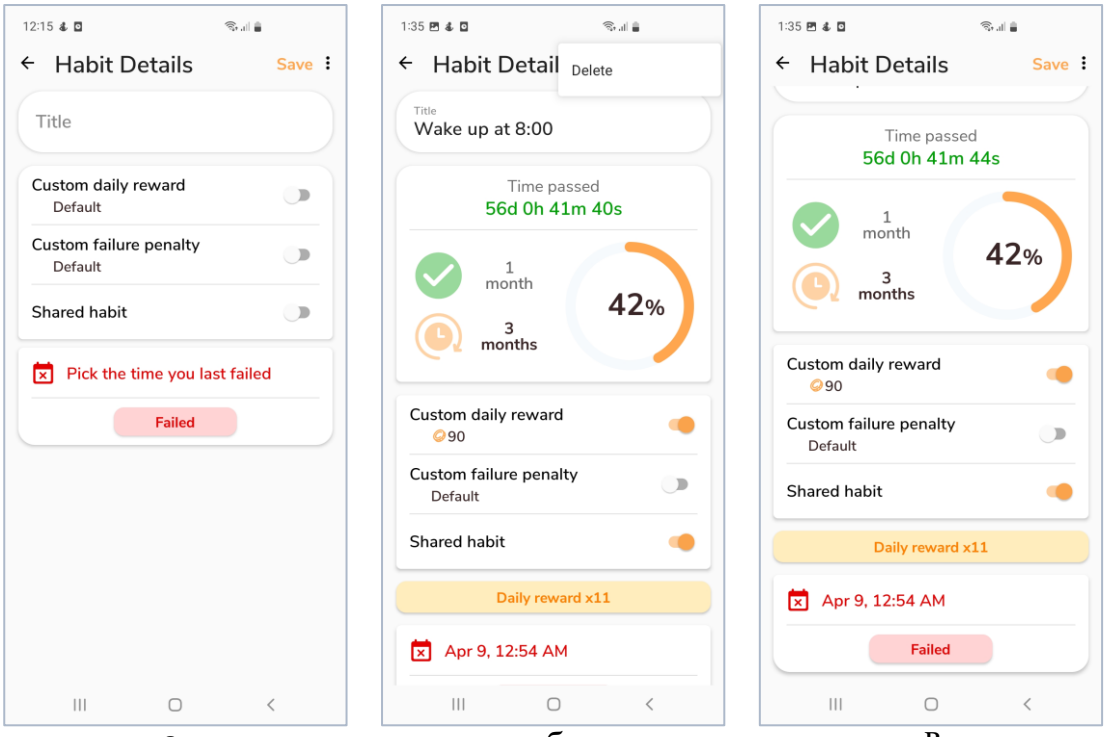


а б

Рис. 13 - Вікно зі звичками (а - пусте вікно, б - вікно з даними)

На вікні створення звички (Рис. 14а) є 2 обов’язкових поля (назва звички та остання дата невдачі) та 3 необов’язкових поля (налаштування щоденної винагороди, штрафу за невдачу та налаштування видимості звички для друзів (буде далі)). Останнє налаштування є тільки в онлайн режимі застосунку. Якщо користувач спробує створити звичку без назви,

або без визначення останньої дати невдачі, відповідні повідомлення з помилкою будуть виведені на екран. Вікно редагування звички (Рис. 14б,в) має усі ті ж поля, що і вікно створення звички (в коді це один фрагмент), та інші елементи: панель з інформацією про час утримання від/дотримання звички та кнопка з щоденним бонусом (працює так само, як і на головному вікні звичок). Натиснувши на кнопку “Failed”, користувачу по-черзі відкриється два діалогових вікна для вибору дати і часу останньої невдачі. Усі зміни над звичкою, окрім забирання щоденної винагороди, зберігаються тільки після натиснення на кнопку “Save”. За допомогою контекстного меню (Рис. 14б) користувач може видалити звичку.

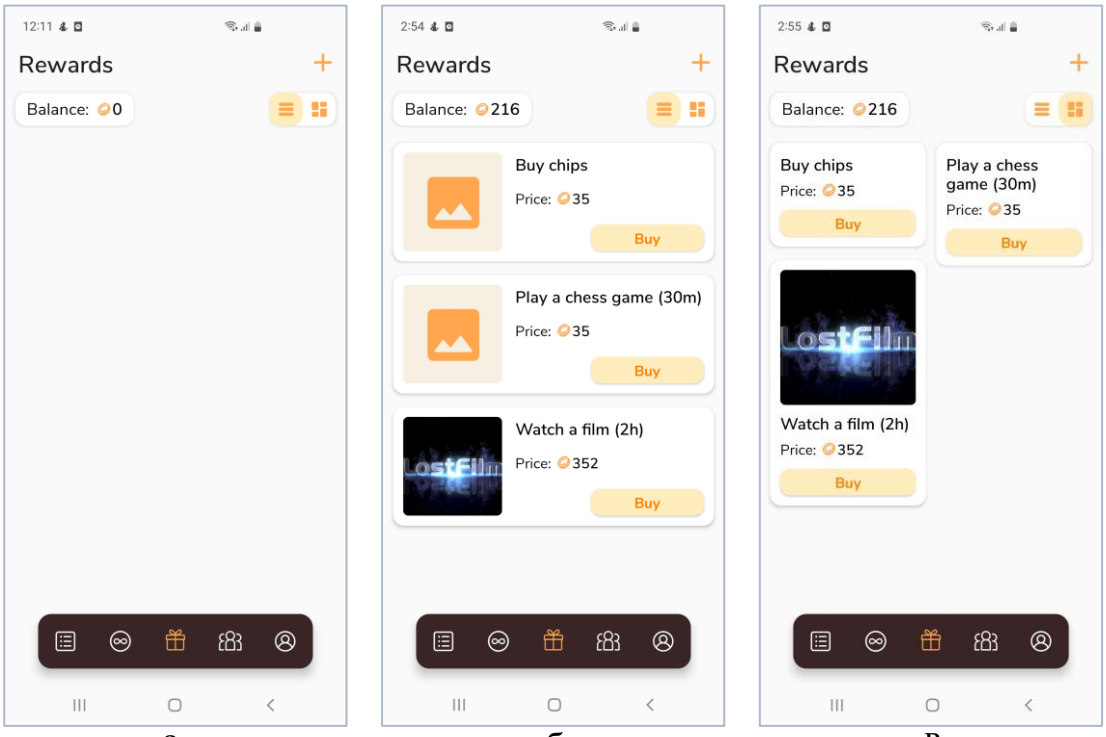


а б в

Рис. 14 - Вікно створення/редагування звичок (а - створення звички, б - редагування звички та контекстне меню, в – редагування звички)

Третій пункт в навігаційному меню – вікно зі списком винагород (Рис. 15). На ньому користувач може побачити свій баланс кредитів, список усіх винагород і також переключатись між двома способами

відображення списку винагород: лінійне (Рис. 15б) та сітчасте (Рис. 15в) (використані LinearLayoutManager та StaggeredGridLayoutManager). Вибраний спосіб відображення зберігається при наступних відкриттях даного вікна, незалежно від режиму застосунку (за допомогою DataStore). Так само як і в інших вікнах, натиснувши на кнопку “плюс” або на конкретну винагороду зі списку, користувач перейде до вікна створення/редагування винагороди (Рис. 16а,б). Кожен елемент списку містить наступну інформацію: назву винагороди, її зображення та ціну. А також у кожної винагороди є кнопка “Buy”, натиснувши на яку користувач може перейти до діалогового вікна покупки (Рис. 16в).



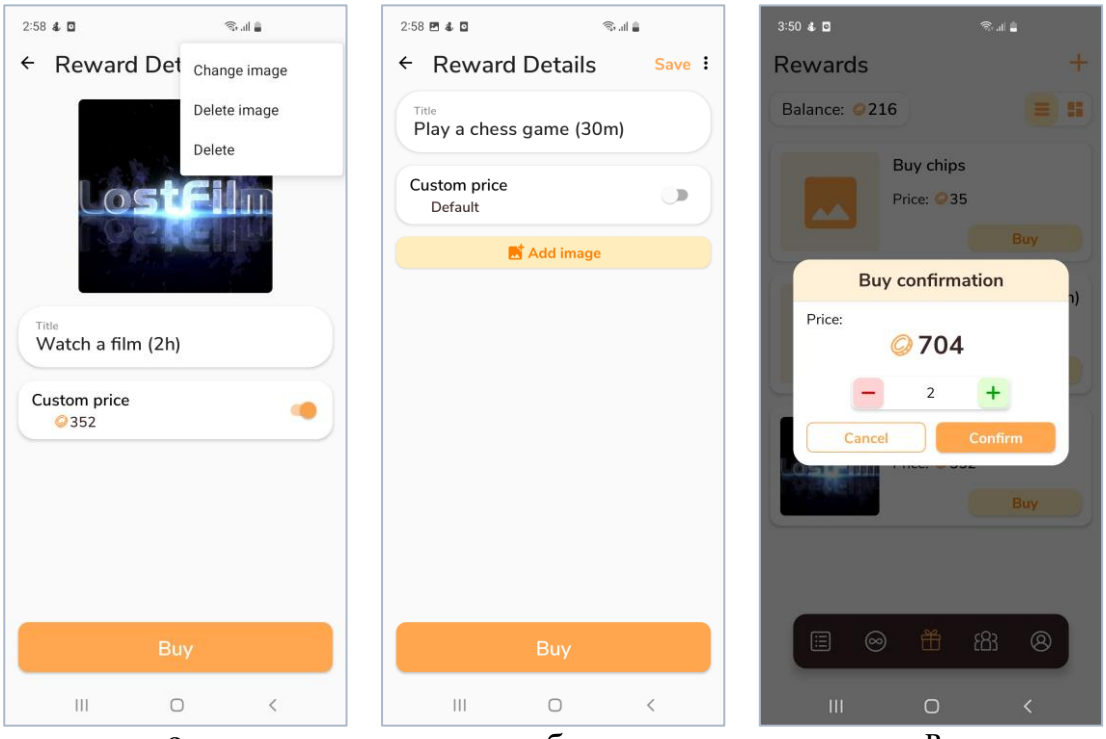
а б в

Рис. 15 - Вікно з винагородами (а - пусте вікно, б - лінійне розташування винагород, в - сітчасте розташування винагород)

На вікні створення/редагування винагороди (Рис. 16а,б) є два поля: назва (обов’язкова) та ціна (необов’язкова). Якщо спробувати зберегти винагороду без назви, буде виведене відповідне повідомлення з помилкою. Якщо винагорода має свою картинку, вона буде відображатись зверху

вікна, а також в контекстному меню будуть доступні 3 опції (Рис. 16а): зміна або видалення зображення та видалення винагороди. Інакше, якщо картинки у винагороди немає, для користувача буде доступна кнопка для вибору зображення, а за допомогою контекстного меню користувач зможе тільки видалити винагороду. Знизу вікна розташована кнопка, натиснувши на яку відкриється діалогове вікно покупки товару (Рис. 16в).

За допомогою цього вікна користувач може купити від 1 до 1000 винагород одного виду за раз. Ціна змінюється відповідно вибраної кількості одиниць товару. Якщо на балансі у користувача недостатньо кредитів для здійснення покупки, відповідне повідомлення буде виведено на екран.



а б в

Рис. 16 - Вікна створення/редагування винагороди (а - винагорода із зображенням та контекстним меню, б - винагорода без зображення) та вікно підтвердження покупки винагороди (в)

Четвертий пункт навігаційного меню в офлайн режимі – це головне вікно налаштувань застосунку (Рис. 17). В онлайн режимі перейти до

цього екрану можна, натиснувши на кнопку налаштувань на панелі інструментів вікна особистого кабінету (Рис. 20а). З головного вікна налаштувань користувач може перейти до загальних налаштувань задач (Рис. 18а), звичок (Рис. 18б) та винагород (Рис. 18в), а також вийти з акаунту (якщо це онлайн режим) та вийти з офлайн режиму.

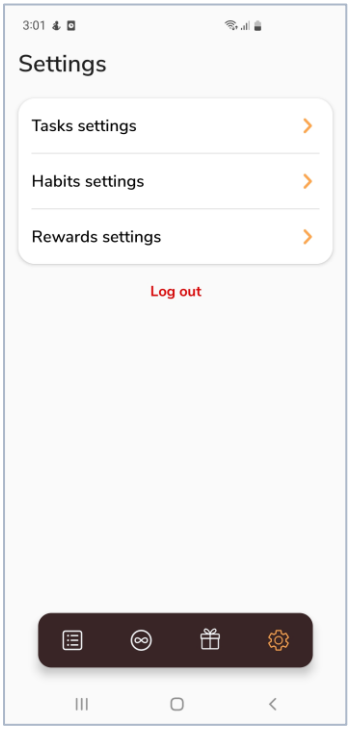
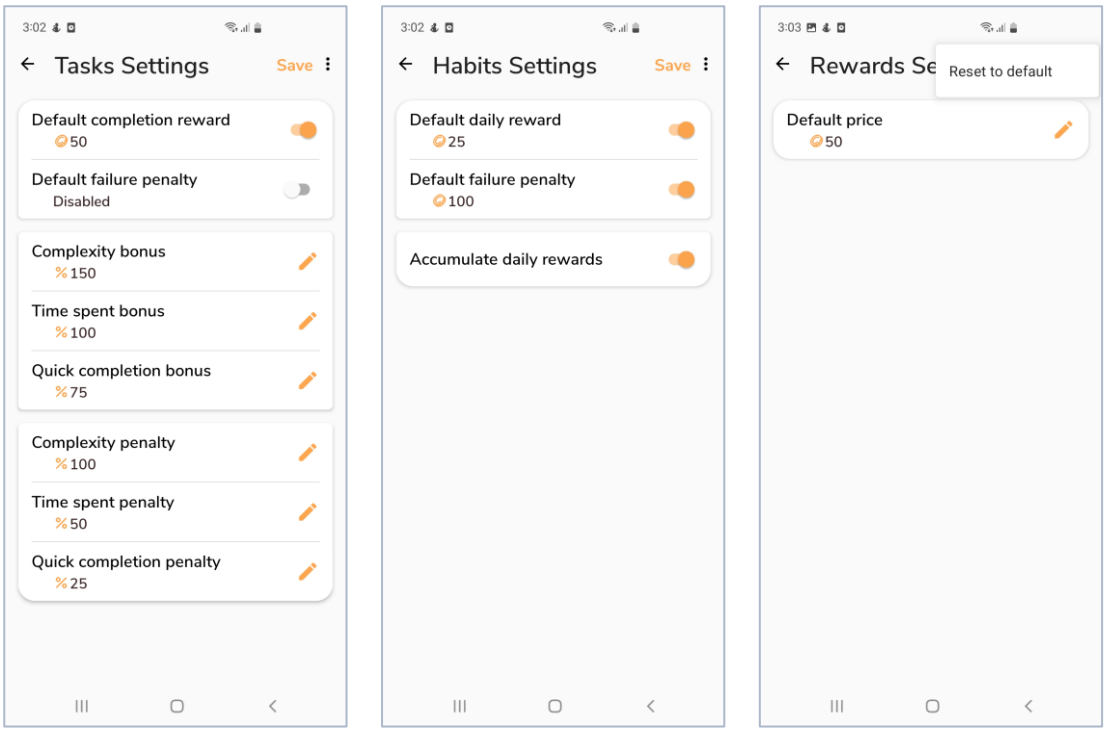


Рис. 17 - Головне вікно налаштувань

Використання усіх налаштувань на трьох вікнах (Рис. 18) було описано раніше. Стандартні значення для винагород та штрафів (або ціни винагород) використовуються в тому випадку, якщо для окремих задач/звичок/винагород не вказані особисті значення. Додаткові бонуси та штрафи за трьома параметрами для задач (Рис. 18а) можуть бути використані при виконанні задачі користувачем у тому разі, якщо ці параметри виявились неправильно спрогнозованими при визначенні суми кредитів у якості винагороди/штрафу. Налаштування з акумулюванням щоденних бонусів за дотримання умов звички (Рис. 18б) визначає, чи

будуть бонуси накопичуватись, чи користувач повинен щоденно забирати бонус (тільки після чого наступні бонуси почнуть накопичуватись).

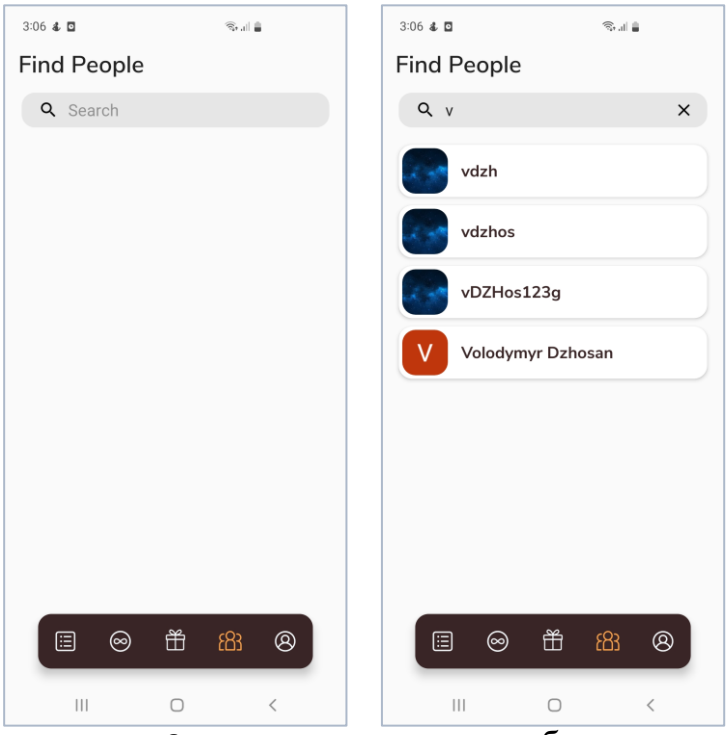


а б в

Рис. 18 - Вікна загальних налаштувань (а - задач, б - звичок, в - винагород)

Усі наступні вікна доступні для користувача лише в онлайн режимі. Четвертий пункт навігаційного меню в онлайн режимі – вікно пошуку інших користувачів (Рис. 19). Коли користувач вводить якийсь запит в поле для пошуку, інші користувачі з нікнеймами, які починаються на цей текст (незалежно від регістру), з’являються в списку нижче. Такий пошук відбувається кожен раз після зміни запиту. Для того, щоб не заставляти користувача довго чекати, поки велика кількість інших акаунтів загрузиться, використаний принцип пагінації (pagination). Тобто після кожної зміни запиту, підвантажуються лише перші 15 користувачів. І тільки при пролистуванні списку вниз, буде виконуватись пошук наступних акаунтів партіями по 15 штук. В коді даний фрагмент використовується і для відображення вікон з друзями та запитами на

дружбу (Рис. 20б,в; 22б). Проте у цих випадках пагінація не використовується через обмеження в складності запитів до бази даних Firestore. Усі друзі/запити відразу підтягуються з бази даних та тільки фільтруються при зміні запиту в полі пошуку. Рішення зробити це саме таким (на перший погляд неефективним) способом було прийняте, оскільки у застосунку з такою сферою застосування імовірніше за все, що у користувачів зазвичай буде невелика кількість друзів та запитів.



а б

Рис. 19 - Вікно пошуку користувачів (а - без запиту, б - із запитом)

Останній (п’ятий) пункт навігаційного меню в онлайн режимі – особистий кабінет користувача (Рис. 20а). Дане вікно використовується для відображення як особистого акаунту (Рис. 20а), так і інших користувачів (Рис. 21; 22а).

Якщо це особистий кабінет, на вікні можна побачити:

- Аватарку користувача, яка підтягується з Google акаунту при реєстрації.

- Нікнейм користувача під аватаркою, який підтягується з Google акаунту при реєстрації. Нікнейм можна копіювати, натиснувши на нього, для зручності користувачів ділитись своїм акаунтом.
- Дві кнопки для переходу до вікна зі списками друзів (Рис. 20б) або запитів на дружбу (Рис. 20в).
- Список усіх звичок користувача, у кожної з яких є перемикач, який змінює їх видимість для друзів. Ділитись своїми звичками може бути корисно для того, щоб друзі бачили прогрес один одного, що може добавити мотивації досягати своїх цілей.

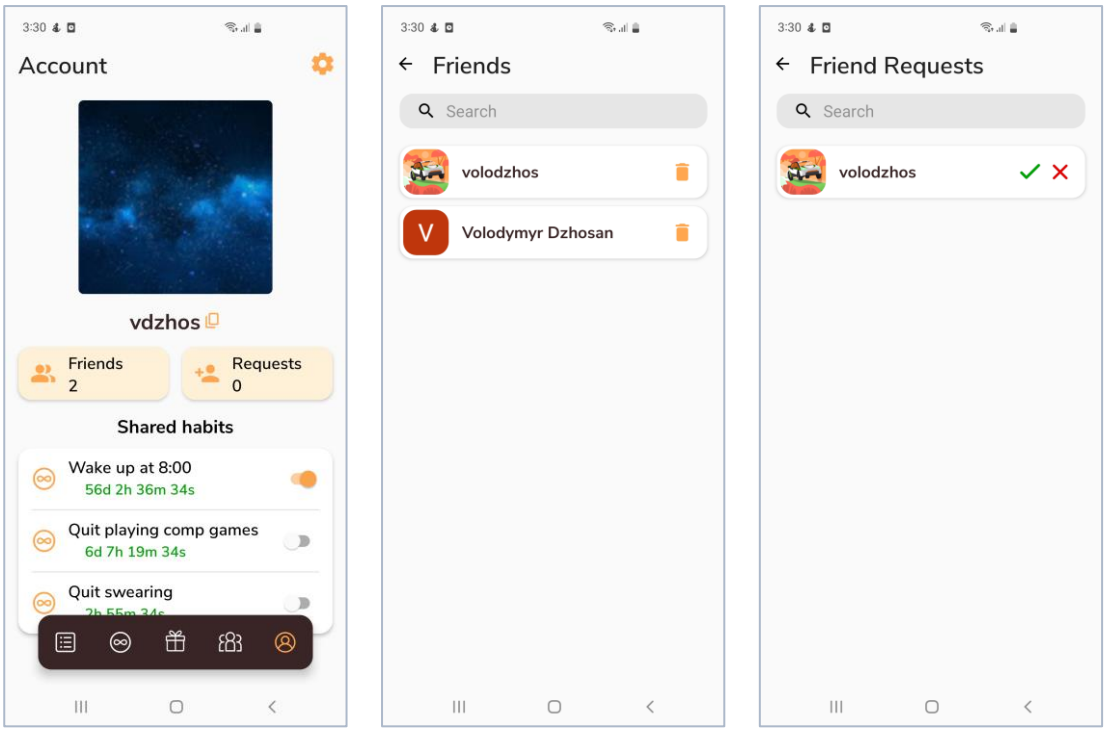
Стосовно вікон зі списками своїх друзів та запитів на дружбу, на кожному елементі з цих списків (які позначають акаунти інших користувачів) присутні кнопки. Якщо це список друзів – одна кнопка, за допомогою якої можна видалити друга (Рис. 20б), а якщо це список запитів на дружбу – дві кнопки, одна з яких для прийняття запиту, а інша для його відхилення (Рис. 20в).

Якщо це сторінка іншого користувача, на вікні можна побачити:

- Аватарку користувача, яка підтягується з Google акаунту при реєстрації.
- Нікнейм користувача, як заголовок сторінки (Рис. 21; 22а), який підтягується з Google акаунту при реєстрації.
- Кнопку для переходу до вікна зі списком друзів іншого користувача (Рис. 22б).
- Різні кнопки, в залежності від того, хто користувач для нас:
    - Ніхто – кнопка, за допомогою якої можна відправити запит на дружбу (Рис. 21а).
    - Користувач, якому ми надіслали запит на дружбу – кнопка, яка скасовує надісланий запит на дружбу (Рис. 21б).

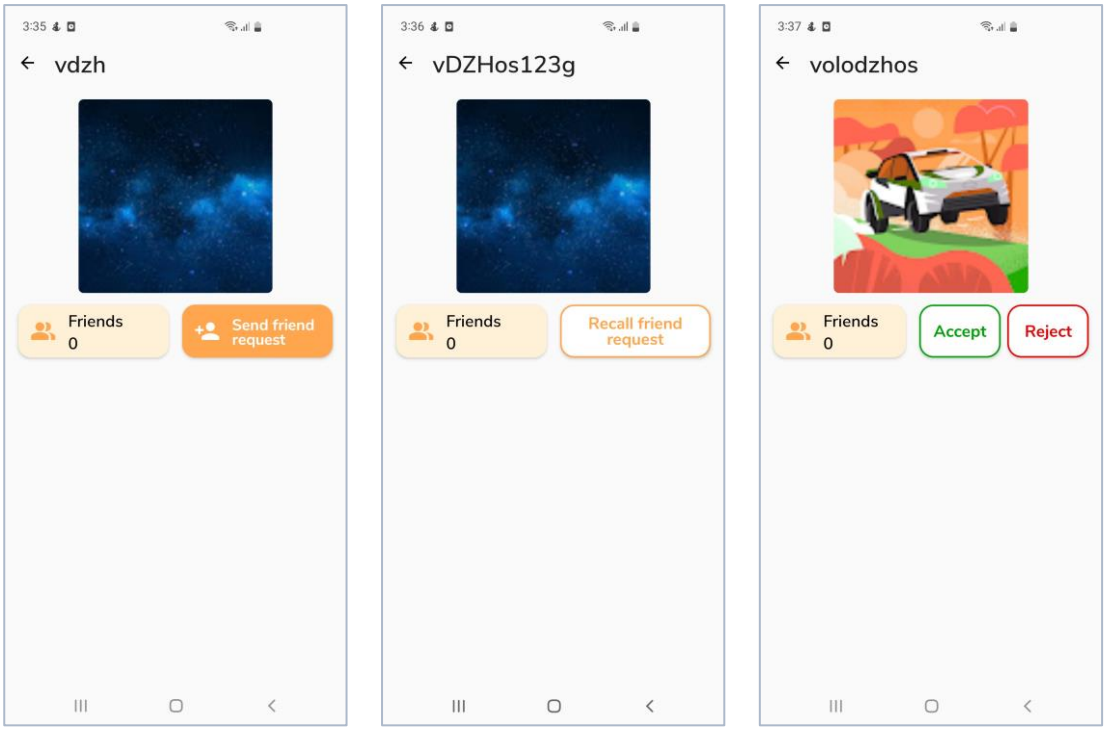
- Користувач, який надіслав нам запит на дружбу – дві кнопки. Перша – прийняття запита на дружбу, друга – його відхилення (Рис. 21в).
- Друг – кнопка, за допомогою якої можна видалити користувача з друзів (Рис. 22а).

- У випадку, якщо інший користувач - друг, список його звичок, якими він ділиться з іншими друзями (Рис. 22а).



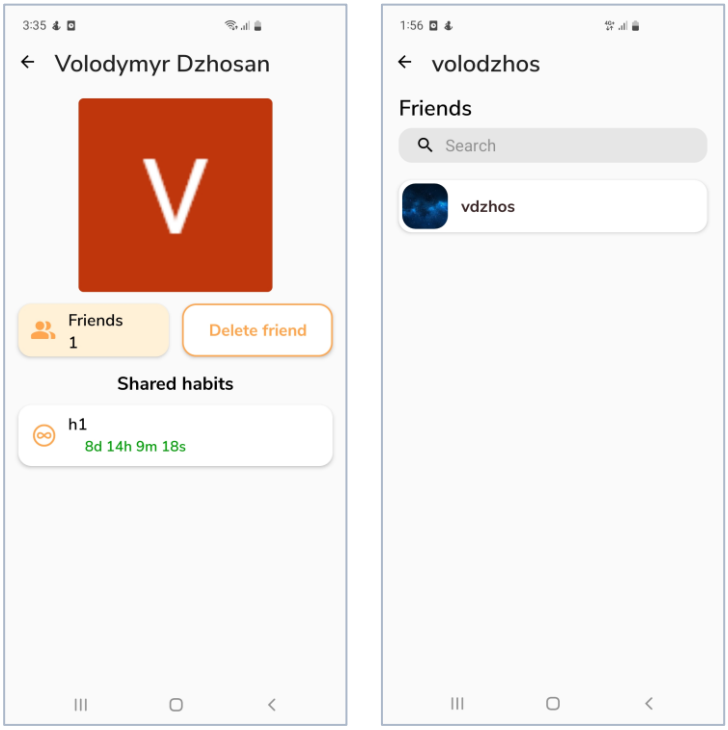
а б в

Рис. 20 – Вікно особистого кабінету (а), вікно пошуку свої друзів (б), вікно пошуку своїх запитів на дружбу (в)



а б в

Рис. 21 - Вікно з користувачами різних статусів (а - відправлення запиту, б - скасування запиту, в - прийняття/відхилення запиту)



а б

Рис. 22 - Вікно з користувачем-другом (а) та вікно з друзями інших користувачів (б)

Висновок

У результаті виконання курсової роботи був розроблений застосунок, який стане в нагоді людям, які хочуть розвинути в собі таку важливу якість як дисциплінованість.

Звісно, в майбутньому додаток можна продовжити удосконалювати: розширити функціонал в плані роботи основних елементів застосунку (задач/звичок/винагород) та додати більше можливостей для взаємодії користувачів. Проте уже за наявного функціоналу, застосунок може бути зручним на надійним інструментом для досягання своїх цілей.

Список використаних джерел

1. Recommended app architecture – [Електронний ресурс]. – Доступно з: https://developer.android.com/topic/architecture#recommended-app-arch
2. MVVM Architecture – [Електронний ресурс]. – Доступно з: https://blog.mindorks.com/mvvm-architecture-android-tutorial-for-beginners-step-by-step-guide
3. App development platform – Firebase – [Електронний ресурс]. – Доступно з: https://firebase.google.com/
4. Firebase Authentication – [Електронний ресурс]. – Доступно з: https://firebase.google.com/docs/auth
5. Cloud Firestore – [Електронний ресурс]. – Доступно з: https://firebase.google.com/docs/firestore
6. Cloud Storage for Firebase – [Електронний ресурс]. – Доступно з: https://firebase.google.com/docs/storage
7. Save data in a local database using Room – [Електронний ресурс]. – Доступно з: https://developer.android.com/training/data-storage/room
8. Using Dagger in Android apps – [Електронний ресурс]. – Доступно з: https://developer.android.com/training/dependency-injection/dagger-android
9. Navigation component – [Електронний ресурс]. – Доступно з: https://developer.android.com/guide/navigation
10. Glide library documentation – [Електронний ресурс]. – Доступно з: https://github.com/bumptech/glide#glide
11. Compressor library documentation – [Електронний ресурс]. – Доступно з: https://github.com/zetbaitsu/Compressor#compressor
12. DataStore – [Електронний ресурс]. – Доступно з: https://developer.android.com/topic/libraries/architecture/datastore

13. Online-service – Figma – [Електронний ресурс]. – Доступно з: https://www.figma.com/